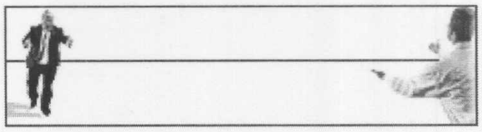
NEWS & ANALYSIS
@embeddedtechnology.com



Features

Return to News & Analysis Home

Embedded TCP/IP: Grasping The Fundamentals

7/19/2000 *By Tracy Thomas, Software Engineer, U S Software Corporation*

There is a wealth of reference material on the subject of networking and the TCP/IP protocols. It is often difficult for the embedded systems programmer to sift through all of this information to find the useful parts. Yet, an increasing number of embedded applications benefit from the addition of networking capability. This article is a practical guide to the basics of TCP/IP and contains the information most often misunderstood by programmers who are new to networking.

The first point to understand is that when we refer to TCP/IP, we often mean the family of protocols which include TCP (Transport Control Protocol), UDP (User Datagram Protocol), IP (Internet Protocol), and some underlying link layer such as Ethernet. Data sent using the TCP protocol is referred to as a segment and data sent using the UDP protocol is referred to as a packet. IP is the network layer that lies under TCP and UDP. IP provides unreliable, connectionless packet delivery to a specified host address. IP packets, called datagrams, can be lost, duplicated, or delivered out of order. The advantage of IP is that an IP datagram provides a universal method for delivering data, independent of the underlying network technology.

UDP and TCP sit on top of IP at the transport layer. Both UDP and TCP use port numbers to demulitplex data sent to a host. A port number is specific to an application; the use of multiple port numbers allows a single host to run multiple networking applications. Each UDP packet and TCP segment has a source and destination port number. A host that waits for incoming connections is referred to as the server, and the host that initiates a connection is referred to as the client. Servers "listen" on well-known port numbers for common applications such as FTP (File Transfer Protocol), Email, and HTTP. Clients generally choose a random source port number and connect to a server at a well-known port. Custom applications should use a port number greater than 1024, because port numbers below 1024 are reserved for well-known applications.

UDP provides best effort delivery of data with an optional checksum to preserve data integrity. UDP packets have the same reliability as IP; packets are not guaranteed to be received in order at the remote host. TCP provides a reliable stream of data by using sequence and acknowledge numbers to recover lost data, detect out-of-order segments, and resolve transmission errors. As a result of providing this reliability, the TCP protocol is more complicated than the UDP protocol.

TCP is connection-oriented while UDP is connectionless. This implies that, to send a UDP packet, a client addresses a packet to a remote host and sends it without any preliminary contact to determine if the host is ready for data. If a client sends a UDP packet to a host that is not listening on the destination port, the host returns an ICMP (Internet Control Message Protocol) error. The remote

the host returns an ICMP (Internet Control Message Protocol) error. The remote host can send back this error message as long as it is running a TCP/IP stack capable of processing incoming datagrams to the ICMP layer. Thus, the client has some notification if the remote host is not accepting data, but this notification is limited, and the client receives no confirmation that the data was received. Because of its lack of a reliability mechanism, UDP also has no throttling mechanism. UDP packets can be sent at full speed – as fast as the underlying physical device can send them.

On slow processors, UDP's lack of overhead can make a large difference in the throughput when compared to TCP. On fast processors, the difference is not as large. The lack of an end-to-end connection in UDP means that it can be used to send one-to-many and many-to-many type messages. This feature is used to send broadcast and multicast messages. One example of the use of UDP in broadcast messages is the Dynamic Host Control Protocol, which uses a broadcast message at system boot time to find a DHCP server to supply network configuration information.

TCP provides a connection-oriented, reliable stream of data. Before sending data to a remote host, a TCP connection must be established. This means that only data between two end hosts may be exchanged over a TCP connection. Establishing a TCP connection involves a "three-way handshake":

1. Client sends SYN segment to server to request connection
2. Server sends client SYN-ACK segment to a) request connection and b) acknowledge the client's connection request segment
3. Client sends ACK to server to acknowledge server's connection request segment.

Every TCP segment that contains data is acknowledged to provide reliability. The acknowledge segments themselves are not acknowledged to prevent an infinite recursion. When a connection is requested, the client randomly picks an initial sequence number. In step 2 of the connection establishment, the server picks its own initial sequence number, and acknowledges the client's sequence number in its acknowledge number. Thus, every TCP segment contains a sequence number which acts as a placeholder in the data stream, and an acknowledge number to notify the remote host of reception of its data.

Transmission errors are resolved by keeping each data segment until it has been acknowledged. If the data is not acknowledged in a set amount of time, the client retransmits the data to the remote host. If the data is still not acknowledged, the client keeps retransmitting the segment with successively larger time intervals between transmissions. After a set time with no acknowledge, the connection will no longer be usable, and an error condition will be returned to the application. A TCP connection can remain open indefinitely. If no data is sent on an open connection, the connection remains open.

The only way for a host to see if the remote side is still there is to send data. A TCP connection remains open until it is closed by both hosts. The TCP close sequence requires four steps:

1. The client initiates the close by sending a FIN segment to the server. At this point, the client may no longer send data to the remote host, but it can still receive data.
2. The server sends an ACK segment to the client to acknowledge the FIN segment. Although it is not common, the server may still send data to the client. This condition is called the "half-close".
3. The server sends a FIN segment to the client to close its connection. Now, the server may no longer send any data.

4. The client sends an ACK segment to acknowledge the server's FIN segment.

After step 4, the connection is closed. However, the client must not reuse this connection for a timeout period called the 2MSL timeout. This delay attempts to ensure that a subsequent connection will not be confused with the original connection. In the world of enterprise networking, this timeout is on the order of two minutes, and does not have much impact on the client because the client system has a large amount of resources available for creating new connections. In an embedded system, two minutes is equivalent to an eternity and resources are often limited. Therefore, in embedded TCP/IP implementations, the 2MSL timeout is often reduced from the two-minute value.

TCP regulates the data rate, or throughput, by the use of a sliding window. With TCP, a client cannot establish a connection and blast the remote host with data at full speed. The data flow is regulated by the requirement for acknowledgement of data and by the TCP window. In each TCP segment, the host advertises the amount of data it is ready to accept – this amount is the TCP window.

The sliding window refers to the progression of data sequence numbers as data is sent throughout a connection. The window size advertised by the remote host specifies the maximum amount of data the sender can have outstanding (unacknowledged). Data to the left of the current window is in the past, and cannot be resent. Data to the right of the current window cannot be sent until the window moves. So, only data within the sliding window may be sent to the remote host. The sliding window moves to the right only when a data segment is acknowledged. Thus, the acknowledgement of data along with the window keeps TCP from sending data too fast. For example, in Fig. 1, the client can send segments 7, 8, and 9, and segments 4, 5, and 6 are waiting for acknowledgement. If we receive an acknowledgement for segment 4, and the window slides one place to the right, then we can now send segment 10. This concept of the sliding window is also referred to as flow control and limits the amount of traffic one host may place on the network to the amount that the remote host is actually processing. The remote host can close its window (advertise zero bytes can be sent). If the remote window is closed, the client can no longer send data to the remote host, but must instead probe the remote host to find out when the window opens.

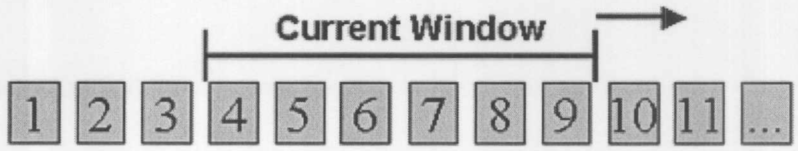


Figure 1: TCP Sliding Window

One important feature of TCP to understand is that the data is delivered as a stream. If the client application does ten 128 byte writes, there is no way to know if the underlying TCP will deliver ten 128 byte segments, or one 1280 byte segment. The important point for application developers is that an additional protocol often needs to be used on top of TCP to mark the beginning, end, and type of data. FTP, Telnet, and Email (SMTP) are common application examples that place additional protocol headers on top of TCP. When you design your own application with TCP, you will need some way to break up the stream of data into recognizable parts.

So, we can now see that TCP's reliability comes at the price of being much more complicated than UDP. The embedded programmer needs to understand these basic concepts of TCP and UDP to design the best application for the job at hand. The main points to keep in mind are that UDP is connectionless, unreliable, and has no throttle on its throughput. TCP offers a reliable data stream at the price of more processing overhead and reduced throughput. Embedded designers have many different concerns – for some code size is of ultimate importance, while for others, data rate and reliability are a higher priority. The simplicity of UDP is slightly misleading – UDP applications typically require code at the application layer to build in some level of reliability. Data sequencing is the most common addition because it allows the application to mark missing data and to resolve out-of-order packets.

Some example applications illustrate the merits of using UDP and TCP for different needs. A central monitor tracking the status of sensors throughout a building for temperature and humidity control receives data from each sensor once per minute. This application could use either UDP or TCP. If an individual report can be dropped with minimal consequences, then UDP can be used. If the sensor data needs to be tracked in a database and each reading is important, then TCP should be used. In this example, reliability guides the choice. Restricting the protocol to UDP may allow this type of application to fit on a minimal microprocessor that could otherwise not be network-enabled. Video-conferencing and Voice over IP are ideal examples for using UDP. Throughput is very important in the real-time reception of audio and video. By the time a reliable protocol could retransmit a packet, the moment to play that packet has passed. However, these types of applications require a method to track data sent and received, so an application layer protocol is layered on top of UDP to provide sequencing of data. TCP is appropriate for applications that require reliability in data transmission. Data acquisition and control applications are examples. Common applications such as email and the Web use TCP as well.

Embedded applications benefit greatly from standardized network protocols. With minimal resources, an embedded device can be monitored and controlled remotely. The points covered in this article are only a brief introduction to some of the issues involved in using the TCP/IP protocols. The embedded programmer should consider the strong and weak points of the networking protocols before choosing the appropriate set for their design.

Contributed by U.S. Software Corporation, 7175 NW Evergreen Parkway, Suite 100, Hillsboro, OR 97124. Tel: (503) 844-6614; Fax: (503) 844-6480.

References:

Postel, J. B., ed. 1981. "Transmission Control Protocol" RFC 793, (Sept).
Stevens, W. R. 1994. *TCP/IP Illustrated, Volume 1*. Addison-Wesley, Massachusetts
Comer, D. E. 1995. *Internetworking with TCP/IP Volume 1*. Prentice Hall, New Jersey

Forward This Article To An Associate

Enter all or part of a topic name in the form

Search!

Go to **Advanced Search** (with **Editorial Archived News**)